for example, all managers attending an upcoming meeting at 10 AM, the classifier can anticipate and have prepared for delivery, or even deliver to the each manager via e-mail, an RSD file that reflects the existing state of the database such that all managers are working from the same "copy" of the database. These are but a few of the variations that can be employed in accordance with the disclosed architecture.

[**0048**] As indicated herein, the disclosed RSD schema also supports segmentation, where multiple physical units (files/resources) can be segmented and then combined into a single logical unit. Segmentation is designed to satisfy particular extension scenarios and to enable re-use of modular RSD components in multiple applications, as well as improving readability and manageability of the file. In some cases, the capability to segment files may impact the validation of the RSD file, particularly in the case of name references to structures which can be stored in another file segment.

[**0049**] Referring now to **FIG. 6**, there is illustrated a block diagram of a distributed system **600** where multiple relational databases that comprise an overall database each have respective RSD files that when retrieved can be combined to provide a more comprehensive view of the overall database. There is provided a first database **602** and associated first RSD file **604**, a second database **606** and associated second RSD file **608** and, a third database **610** and associated third RSD file **612**. The databases (**602**, **606**, and **610**), disposed on a wired or wireless network **614**, can be accessed by a client **616**, such that the client **616** retrieves the one or more of the respective RSD files (**604**, **608**, and **612**) for processing either online or offline. Here, the client **616** has retrieved the first RSD file **604** and the second RSD file **608** for use in further processing. The client **616** can process these files (**604** and **608**) separately, or combine the files (**604** and **608**) into a single RSD file **618** for overall processing.

[**0050**] If the user has made any changes to the single file **618**, this can be propagated back to the respective relational databases whereby the single file **618** is segmented back into its constituent files (RSD file **604** and RSD file **608**) and transmitted separately back to respective databases (**602** and **606**) for merging thereinto. It is further to be appreciated that only the RSD file that incorporates the changes may need to be transmitted back for merger with the existing database.

[**0051**] The client **616** may also be running separate applications such that each application runs only one of the data files. For example, if the client were running two different applications, a first application could retrieve and process the first RSD file **604**, and the second application could retrieve and run the second RSD file **608**. Thus, the two RSD files (**604** and **608**) need not be combined for processing at the client. Of course, in a disconnected environment, the RSD files (**604** and **608**) could be stored on the client **616** for later processing, and then uploaded to the respective databases (**602** and **606**) when reconnected to the network **614**.

[**0052**] Note that RSD can store much more than a single database, where database is defined as a "catalog" in the ANSI SQL standard. The RSD by itself can store one to many catalogs (essentially separate schemas that are accessible via the same application connection) that make up a single database instance. However, when combined with a DataLocationPolicy concept, the RSD can expose a "logical instance" which transcends the boundaries of a single physical server and allows the RSD to represent an entire network of database instances, where logical database structures (tables/rows) are bound to physical locations (the server/catalog/schema/table) at runtime based on application logic and/or a policy file.

[**0053**] RSD Language

[**0054**] Where mapping is concerned, the RSD schema is a logical view of the metadata required to perform efficient mappings and provides the following: sufficient metadata about the relational domain to allow the CQR (common query runtime) engines to efficiently generate and/or execute CRUD (Create, Read, Update, and Delete) operations against the database based on actions performed in the target domain; easily readable; easily editable; capability to segment the RSD file to improve manageability and allow logical extensions to the relational schema; and capability to describe any ANSI relational database. When a query is presented through, e.g., XML, the CQR engine compiles the query into QIL (Query Intermediate Language), optimizes it, and generates SQL statements that can be run against the database.

[**0055**] Since the database name, schema name, and structure names are already separated in the RSD format by XML tags, RSD does not require (or allow) SQL escaping of names in the file format. This means that if names are specified with bracket-escaping, the brackets will be treated as literals in the structure name. In order to prevent potential security holes, RSD users should escape the names of all identifiers in the RSD file. However, this is an implementation detail, in that a generic name validation engine can be built that could accept platform-specific rule sets to describe the name escaping rules of a particular platform. If multiple backends are used, the capability to plug in an escaping/validation module to the compilation process is provided.

[**0056**] Since RSD is intended to be database independent, the RSD architecture only imposes naming constraints to ensure that name references within the CQR framework are unambiguous. RSD identifier names are escaped in name references (but not in declarations) in the following cases: where a "." is present anywhere in the identifier; where it starts with a "$" character; and a "" (white space) is present (in relationships only).

[**0057**] With respect to name uniqueness in RSD, structure identifier names (e.g., Tables, Custom Tables, Views, StoredProcedures, and UserDefinedFunctions) belong to the same namespace and must be globally unique within the logical RSD (the union of all the physical segments). The Custom Tables mapping feature is described in greater detail hereinbelow. Note, however, that these are implementation details specific to SQL Server. These name validations could follow different rules in a different database management system.

[**0058**] For validation specific to SQL Server, the following rules apply. Structures (e.g., Tables, Views, UserDefinedFunctions, StoredProcedures, and CustomTables) have a unique 3-part name within the scope of the logical RSD. For stored procedure naming in SQL Server, on file generation, the number for the stored procedure is concatenated to the name of the procedure using the canonical SQL format spName;number. On file generation, if the number is "1", the generating code can omit the semi-colon and the number.